Fitting algorithm parallelization in ACTS FW Performance Analysis

Xiaocong Ai, Georgiana Mania, Nick Styles
Agenda

- Define the experiments
- Time analysis
  - Fitter parallelization
  - Overall parallelization
- Memory footprint
- Time-memory tradeoff
- Conclusions
- Future work
Define the experiments

Goal

- Investigate the effects of thread parallelization on ACTS code
  - Use the fitting step of the particle reconstruction code for the investigation due to its (relative) simplicity
  - Use Intel TBB as parallelization library
  - Measure and analyse:
    - the execution time and
    - the memory footprint
  - Evaluate the parallelization using two metrics: speedup and efficiency
Define the experiments

Define the environment

- Test scenario:
  - Track fitting of 1 and 10 events, 10k particles per event, different B fields
  - Parallelization points:
    - Events loop (outer loop)
    - Tracks loop (inner loop) → 2x2 main combinations + several configurations for fully parallel scenario

- Hardware clusters used for measurements:
  - AMD Opteron 6168 (4x 12 cores) @ Uni Hamburg
  - Intel Cori KNL (68 cores), 272 hyper-threads @ Berkeley
  - Intel Cori Haswell (2x 16 cores), 2x 32 hyper-threads @ Berkeley
Define the experiments

*TBB magic*

- TBB uses the notion of “task” rather than “thread” parallelization
- TBB may choose not to run parallel threads (num_threads > the actual loop size)
- the number of threads is set in the outer loop and all the inner loops will use threads from the pre-configured thread pool
  - if only some threads are available, ie the num_threads > number of iterations of the outer loop
  - this may lead to deadlocks or inconsistency due to the fact that each free threads can pick up a task from either outer or inner loop
- tbb::task_arena can be used to enforce a specific number of threads but
  - this induces overhead due to thread isolation
  - num_threads = max (scheduler's init value, number of physical hyper-threads)
- Some tests were run with sequential implementations for Sequencer & Fitter to overcome the above restrictions and to better control the test env
Time analysis – Fitter parallelization

Fitting time analysis

- 1 event x 10k particles, parallel Fitter, test on AMD, Intel KNL and Intel Haswell
- similar values regardless of the B field (same EiggenStepper). Could be improved?
- Quickly hits Amdahl’s law => scalability issues
Time analysis – Fitter parallelization

**Speedup of the parallelization**

- Speedup \( Sp = \frac{Ts}{Tp} \);  
  \( Ts = \) time for the serial execution  
  \( Tp = \) time for the parallel execution

- Good speedup (up to 8x) when \( \text{num\_threads} < \) number of cores

---

**Speedup on AMD**

- No B field  
- Constant Bz = 1T  
- ATLAS B field

**Speedup on Cori KNL node**

- No B field  
- Constant Bz = 1T  
- ATLAS B field

**Speedup on Cori Haswell node**

- No B field  
- Constant Bz = 1T  
- ATLAS B field
Time analysis – Fitter parallelization

Efficiency of the parallelization

- Efficiency $E = \frac{Sp}{p}$; $Sp =$ speedup
  
  $p =$ number of parallel cores or threads

- Good efficiency (up to 90%) when num_threads $\ll$ number of cores
Time analysis – Overall parallelization

Total execution time analysis

- 10 events x 10k particles
- Hardware: AMD vs Intel Cori Haswell
- Total wall clock execution time (incl. I/O) has speedups < 2x and efficiencies < 30%
Time analysis – Overall parallelization

Timing distribution analysis

- 10 events x 10k particles
- Measurements on AMD cluster
- Times extracted from timing.tsv. Is this correct? Normalization needed?
Time analysis – Overall parallelization

Timing distribution analysis

- 10 events x 10k particles, times extracted from timing.tsv
Memory footprint

*Heap profiling*

- 10 events x 10k particles
- Hardware: AMD vs Intel Cori Haswell
- Heap profiler used: Valgrind with massif tool [1]
- Consistent measurements on both clusters
Memory footprint

Heap profiling

- 10 events x 10k particles, **48x1 threads**, max heap : 8075 MB

Images produced with Valgrind + massif tool + massif.js visualizer
Memory footprint

Heap profiling

- 10 events x 10k particles, **1x48 threads**, max heap : 1512 MB

Images produced with Valgrind + massif tool + massif.js visualizer
Time-memory tradeoff

Heap vs processor time

- 10 events x 10k particles, AMD
Time-memory tradeoff

Heap vs processor time

- 10 events x 10k particles, AMD vs Intel Cori Haswell
Time-memory tradeoff

Heap vs processor time with the current configuration

- 10 events x 10k particles, AMD
- ACTS FW master code has a parallel Sequencer and a sequential Fitter
Time-memory tradeoff

Heap vs processor time in the future?

- 10 events x 10k particles, AMD
- ACTS FW master code has a parallel Sequencer and a sequential Fitter
- Suggestion: have parallel implementations for both Sequencer and Fitter

Optimum configuration (naively)
## Conclusions

*Thread number impact on resources*

<table>
<thead>
<tr>
<th>Impact on Increase threads for</th>
<th>Memory</th>
<th>Execution time/event</th>
<th>Total execution time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event loop</td>
<td>Higher peak</td>
<td>No impact</td>
<td>Reduce considerably</td>
</tr>
<tr>
<td>Tracks loop</td>
<td>Little impact</td>
<td>Reduce considerably</td>
<td>Reduce marginally</td>
</tr>
</tbody>
</table>
Future work

Suggestions

- Extends the tests to a range of $p_T$
- Tests different configurations of readers/writers
- Investigate TBB’s options to explore NUMA architectures (processor affinity)
- Use MPI to parallelize the event loop at a process (node) level while keep the thread level parallelization for the tracks loop
- Have an adaptive strategy to choose the optimum thread configuration based on the hardware available at runtime (maybe employ machine learning?)
- Investigate ways to speedup computationally intensive algorithms (through code transpositions, array alignment, loop fusions, etc.)
- Are there other parts of the particle reconstruction code that could benefit (more) from a similar analysis?